Handout #5: Project 1st Module

Due Feb 20th 2023, before 11:59 pm

Submit your report electronically via Gradescope.

General Instructions

- We will be making use of Jupyter notebooks to modularize and visualize the code. Please find the install instructions at the webpage:

  https://jupyter.org/install

  You may also find useful the following resources, which provide tutorials on how to use Jupyter notebooks and the python `numpy` library.

  https://www.codecademy.com/article/how-to-use-jupyter-notebooks

  https://cs231n.github.io/python-numpy-tutorial/

  https://numpy.org/doc/stable/user/quickstart.html

- The project module directory structure is as follows:

```
Project_Module1
├── data
│   ├── regression
│   ├── binary_classification
│   └── fashion-mnist
├── codes
│   ├── Logistic.py
│   ├── MLogistic.py
│   ├── Regression.py
│   └── SVM.py
├── plots
├── utils
└── Notebook_Module1.ipynb
```

You will only need to make changes to the files in the `code` directory and the main `Notebook_Module1.ipynb` notebook. Please do not change the location of any files and maintain the same directory structure. Your generated plots would be stored in the `plots` folder. You do not have to write commands to load any data as that has already been done in the provided `Notebook_Module1.ipynb` file.

- **Important:** Any portions of the code that you must modify start with `YOUR CODE HERE` and end with `END YOUR CODE HERE`. Please do not change any code outside of these blocks.

- We will also make use of the following python libraries: `scipy, scikit-learn, matplotlib`, which can be installed using:

```
pip install -U scikit-learn scipy matplotlib
```

  The code for loading and using these libraries has already been provided in the main file.

- You must upload a pdf of your report to Gradescope by **Monday, 20th February 11:59 p.m.** You may export your Jupyter notebook to a pdf format.

- Include a small section towards the end of your report on the contributions of each group member in the project.

1 Linear Regression

You will work through linear and polynomial regressions. Our data consists of inputs $x_n \in \mathbb{R}$ and targets $y_n \in \mathbb{R}$ for $n \in \{1, \ldots, N\}$ which are related through a target function $y_n = f(x_n)$. Your goal is to learn a linear predictor $h_{\mathbf{w}}(x)$ that best approximates $f(x)$.

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N \tag{1}$$

The main file is the Notebook_Module1.ipynb Jubyter notebook.

(a) (**Visualization**): Visualize the training and test data. What do you observe? For example, can you make an educated guess on the effectiveness of linear regression in predicting the data?

(b) (**Linear Regression**): Recall that linear regression attempts to minimize the objective function

$$J(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}_n) - y_n \right)^2 \tag{2}$$

where $\mathbf{x}_n = (1, x_n)$. In this part we consider linear regression model, where $h_{\mathbf{w}}(\mathbf{x}_n) = \mathbf{w}^T \mathbf{x}_n = w_0 + w_1 x_n$. Note that to take into account the intercept term w_0, we can add an additional *feature* to each instance and set it to one. This is equivalent to adding an additional first column to $\mathbf{X}$ and setting it to all ones.
Modify the method `get_poly_features()` in Regression.py file for the case $m == 1$ to create a matrix $\mathbf{X}$ for linear regression model.

(c) Before tackling the harder problem of training the regression model, complete `predict()` in Regression.py file for the case $m = 1$ to predict $\mathbf{y}$ from $\mathbf{X}$ and $\mathbf{w}$.

(d) Complete the function `loss_and_grad()` to compute the loss function and the gradient of the loss function with respect to $\mathbf{w}$ for a data set $\mathbf{X}$ and targets $\mathbf{y}$ at given weights $\mathbf{w}$. Test your results by running the code in the main file Notebook_Module1.ipynb. If you implement everything correctly, you should get the loss function around 4 and gradient approximately $[-3.2, -10.5]$.

(e) One way to solve linear regression is through gradient descent (GD). Complete the function `train_LR()` to train the linear regression model for given learning rate $\eta = 1e-3$, `batch_size`=30, and number of iterations `num_iters`=10000. Plot the history of the loss function. What is the final value of the loss function and the value of the weight $\mathbf{w}$? Experiment with different learning rates, batch sizes (e.g. full, stochastic), and number of iterations. What do you find works best for producing the best final loss function value?

(f) We can also get a closed form expression to the linear regression. Complete `closed_form()` function in Regression.py to get the optimal weights $\mathbf{w}$. In the main file Notebook_Module1.ipynb to compare the optimal wights $\mathbf{w}$ and loss function obtained from the closed form with the one obtained from GD.

(g) (**Polynomial regression**) Now let us consider the more complicated case of polynomial regression where our hypothesis is:

$$h_{\mathbf{w}}(\mathbf{x}_n) = w_0 + w_1 x_n + w_2 x_n^2 + \ldots + w_m x_n^m \tag{3}$$

Repeat steps $(b) - (f)$ for the general case $m \geq 2$. Note that you only need to modify the `get_poly_features()` in Regression.py file for the case $m \geq 2$. Then, you will use the new features $\mathbf{X}$ in the next steps. Plot the figures and values of the loss function for the case $m = 3$.

(h) (**Overfitting**) For $m = \{1, \ldots, 10\}$, in the main file Notebook_Module1.ipynb, complete the code by using the closed-form solver to determine the best polynomial regression model on the training data. With this model, calculate the loss on both the training data and the test data. Generate a plot depicting how the optimal loss varies with model complexity (polynomial degree). You should generate a single plot with both training and test error. Which degree polynomial would you say best fits the data? Was there evidence of under/overfitting the data? Use your plot to justify your answer.

(i) (**Regularization**) For this problem, we will use ℓ_1-regularization with the previous objective, which lead to the optimization objective:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_n) - y_n)^2 + \lambda ||\mathbf{w}||_1 \tag{4}$$

Modify the method `loss_and_grad()` in the Regression.py file to reflect the effect of ℓ_1-regularization for gradient based approach.

(j) In the main file Notebook_Module1.ipynb, complete the code to find the coefficients that minimize the error for a third-degree polynomial ($m = 3$) given regularization factor $\lambda = \{10^{-8}, 10^{-7}, \ldots, 10^{-2}\}$. Now use these coefficients to calculate the loss function on both the training data and test data as a function of λ using a learning rate of $\eta = 5e - 4$ and batch size of 10. Generate a plot depicting how the loss varies with λ (for your x-axis, let $x = \{1, 2, \ldots, 7\}$ so that λ is on a logistic scale, with regularization increasing as x increases). Which λ value appears to work best?

2 Binary Classification

In this exercise, you will work through a family of binary classifications. Our data consists of inputs $x_n \in \mathbb{R}^{1 \times d}$ and labels $y_n \in \{-1, 1\}$ for $n \in \{1, \ldots, N\}$. We will work on a subset of the Fashion-MNIST dataset which focuses on classifying whether the image is for a *Dress* $(y = 1)$ or a *Shirt* $(y = -1)$. Your goal is to learn a classifier based on linear predictor $h_{\mathbf{w}}(x) = \mathbf{w}^T x$. Let

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^{N \times d}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \{1, -1\}^N \tag{5}$$

The main file is the Notebook_Module1.ipynb Jupyter notebook.

(a) (**Visualization**): Visualize a sample of the training data. What is the dimensions of X_{train}, and X_{test}.

(b) (**Perceptron**): Implement Perceptron Algorithm to classify your training data. Let the maximum number of iterations of the Algorithm $num_{iter} = N$ (number of training samples). At each iteration, compute the percentage of misclassified points in the training dataset, and save it into a `Loss_hist` array. Plot the history of the loss function (`Loss_hist`). What is the final value of the loss function and the squared ℓ_2 norm value of the weight $||\mathbf{w}||_2^2$? Looking at the loss function, can you comment on whether the Perceptron algorithm converges?

(c) (**Perceptron test error**): Compute the percentage of misclassified points in the test data for the trained Perceptron.

(d) (**Logistic Regression**): In this part, we will implement the logistic regression for binary classification. Recall that logistic regression attempts to minimize the objective function

$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\log\left(1+e^{h_{\mathbf{w}}(\mathbf{x}_n)}\right) - \sum_{n=1}^{N}\mathbf{1}_{y_n=1}h_{\mathbf{w}}(\mathbf{x}_n) \tag{6}$$

where $\mathbf{x}_n = (1, x_n)$, and $\mathbf{1}_A = 1$ if A is true and 0 otherwise. Moreover, $h_{\mathbf{w}}(\mathbf{x}_n) = \mathbf{w}^T\mathbf{x}_n$. First, we will add an additional *feature* to each instance and set it to one. This is equivalent to adding an additional first column to $\mathbf{X}$ and setting it to all ones.
Modify the `get_features()` in Logistic.py file to create a matrix $\mathbf{X}$ for logistic regression model [You might use the function `get_poly_features()` from the previous question].

(e) Complete `predict()` in Logistic.py file to predict $\mathbf{y}$ from $\mathbf{X}$ and $\mathbf{w}$.

(f) Complete the function `loss_and_grad()` to compute the loss function and the gradient of the loss function with respect to $\mathbf{w}$ for a data set $\mathbf{X}$ and labels $\mathbf{y}$ at given weights $\mathbf{w}$. Test your results by running the code in the main file Notebook_Module1.ipynb. If you implement everything correctly, you should get the loss function within 0.7 and squared ℓ_2 norm of the gradient around 1.8×10^5.

(g) Complete the function `train_LR()` to train the logistic regression model for given learning rate $\eta = 10^{-6}$, `batch_size` $= 100$, and number of iterations $num_{iters} = 5000$. Plot the history of the loss function (Loss_hist). What is the final value of the loss function and the squared ℓ_2 norm value of the weight $||\mathbf{w}||_2^2$?

(h) (**Logistic Regression test error**): Compute the percentage of misclassified points in the test data for the trained Logistic Regression.

(i) (**Logistic Regression and Batch Size**): Train the Logistic regression model with different batch size $b \in \{1, 50, 100, 200, 300\}$, learning rate $\eta = 10^{-5}$, and number of iterations $num_{iter} = 6000/b$. Train each model 10 times and average the test error for each value of batch size. Plot the test error as a function of the batch size. Which batch size gives the minimum test error?

(j) (**SVM**): In this part, we will implement SVM for binary classification. Recall that SVM attempts to minimize the objective function

$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\max(0, 1 - y_n h_{\mathbf{w}}(\mathbf{x}_n)) \tag{7}$$

where $\mathbf{x}_n = (1, x_n)$, and $\mathbf{1}_A = 1$ if A is true and 0 otherwise. Moreover, $h_{\mathbf{w}}(\mathbf{x}_n) = \mathbf{w}^T \mathbf{x}_n$. First, we will add an additional *feature* to each instance and set it to one. This is equivalent to adding an additional first column to $\mathbf{X}$ and setting it to all ones.
Modify the `get_features()` in SVM.py file to create a matrix $\mathbf{X}$ for SVM model.

(k) Complete `predict()` in SVM.py file to predict $\mathbf{y}$ from $\mathbf{X}$ and $\mathbf{w}$.

(l) Complete the function `loss_and_grad()` in SVM.py to compute the loss function and the gradient of the loss function with respect to $\mathbf{w}$ for a data set $\mathbf{X}$ and labels $\mathbf{y}$ at given weights $\mathbf{w}$. Test your results by running the code in the main file Notebook_Module1.ipynb. If you implement everything correctly, you should get the loss function around 1 and squared ℓ_2 norm of gradient approximately 7.5×10^5.

(m) Complete the function `train_svm()` to train the svm model for given learning rate $\eta = 10^{-6}$, `batch_size` $= 50$, and number of iterations $num_{iters} = 5000$. Plot the history of the loss function (`Loss_hist`). What is the final value of the loss function and the value of squared ℓ_2 norm of the weight $||\mathbf{w}||_2^2$?

(n) (**SVM test error**): Compute the percentage of misclassified points in the test data for the trained SVM.

(o) (**SVM and Batch Size**): Train the SVM model with different batch size $b \in \{1, 50, 100, 200, 300\}$, learning rate $\eta = 10^{-5}$, and number of iterations $num_{iter} = 6000/b$. Train each model 10 times and average the test error for each value of batch size. Plot the test error as a function of the batch size. Which batch size gives the minimum test error?

(p) (**Kernelized SVM**): In this part we will use `sklearn.svm.SVC` library to train a RBF kernelized SVM. RBF kernelized SVM uses a parameter C that represents an inverse ℓ_2-regularization strength (higher C corresponds to a lower regularization). Given the list of possible C values $\{0.01, 0.1, 0.25, 0.75, 1\}$ find the best value by plotting test error vs. C. How is the test error compared to non-kernelized SVM?

3 Multi-Class Classification

In this exercise, you will consider a multi-class image classification task for the Fashion-MNIST dataset [2]. Our data consists of inputs $x_n \in \mathbb{R}^{1\times d}$ and labels $y_n \in \{0, 1, \dots, 9\}$ for $n \in [N]$. Thus:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^{N\times d}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \{0, \dots, 9\}^N \tag{8}$$

The main file is Notebook_Module1.ipynb Jupyter notebook.

(a) (**Visualization**): Visualize a sample of the training data. What is the dimensions of X_{train}, and X_{test}?

(b) (**Multi-class Logistic Regression**): In this part, we will implement the Multi-class logistic regression. Recall that logistic regression attempts to minimize the objective function

$$J(\mathbf{W}) = \frac{1}{N}\left[\sum_{i=1}^{N} \log\left(\sum_{j=1}^{k} e^{w_j^T \mathbf{x}_i}\right) - \sum_{i=1}^{N} w_{y_i}^T \mathbf{x}_i\right] \tag{9}$$

where $\mathbf{x}_n = (1, x_n)$, and $\mathbf{W} = [w_1, \ldots, w_k] \in \mathbb{R}^{k \times d}$ weight matrix, where $w_i \in \mathbb{R}^d$. First, we will add an additional *feature* to each instance and set it to one. This is equivalent to adding an additional first column to $\mathbf{X}$ and setting it to all ones.
Modify the `get_features()` in MLogistic.py file to create a matrix $\mathbf{X}$ for logistic regression model (You may use the function `get_poly_features()` from the previous questions).

(c) Complete `predict()` in MLogistic.py file to predict $\mathbf{y}$ from $\mathbf{X}$ and $\mathbf{W}$. Recall that for a new feature $\mathbf{x}$, we assign $\mathbf{x}$ to a class j^* such that

$$j^* = \arg \max_{j \in \{1, \ldots, k\}} \frac{e^{w_j^T \mathbf{x}}}{\sum_{i=1}^{k} e^{w_i^T \mathbf{x}}} \tag{10}$$

(d) Complete the function `loss_and_grad()` to compute the loss function and the gradient of the loss function with respect to $\mathbf{W}$ for a dataset $\mathbf{X}$ and labels $\mathbf{y}$ at given weights $\mathbf{W}$. Test your results by running the code in the main file Notebook_Module1.ipynb. If you implement everything correctly, you should get the loss function around 2.3 and squared Frobenius norm of gradient matrix approximately 420. Note that $\mathbf{W}$ is a $k \times d$ matrix. Hence, the dimension of the gradient of the function J with respect to $\mathbf{W}$ is a matrix with the same size of $k \times d$.

(e) Complete the function `train_LR()` to train the logistic regression model for given learning rate $\eta = 1e - 7$, `batch_size` $= 200$, and number of iterations $num_{iters} = 1500$. Plot the history of the loss function (Loss_hist). What is the final value of the loss function and the square Frobenius norm of the weight matrix $||\mathbf{W}||_F^2$?

(f) (**Logistic Regression test error**): Compute the percentage of the misclassified points in the test data for the trained Logistic Regression.

(g) (**ℓ_1 Regularization**): In this part, we will use ℓ_1-regularization so that our regularized objective function is

$$J(\mathbf{W}) = \frac{1}{N} \left[\sum_{i=1}^{N} \log \left(\sum_{j=1}^{k} e^{w_j^T \mathbf{x}_i} \right) - \sum_{i=1}^{N} w_{y_i}^T \mathbf{x}_i \right] + \lambda \sum_{j=1}^{k} \sum_{l=1}^{d} |w_{jl}| \tag{11}$$

Modify the function `loss_and_grad()` by adding the effect of regularization.

(h) Consider the allowed set of values of λ as $\Lambda := \{0, 10^{-6}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$.

i. In the main file Notebook_Module1.ipynb, complete the code to train the logistic regression model with regularization parameter $\lambda \in \Lambda$. Now use the obtained final model to find the test error and the final training error as a function of λ. You must train each model using learning rate $1e^{-7}$, `batch_size`=200 and number of iterations 3000. Generate a plot depicting how the test error varies with λ (x-axis represents λ). Which λ value appears to work best?

ii. Now use k-fold cross validation (with $k = 5$) to find a suitable value for the regularization coefficient. Specifically, divide the training dataset (at random) into $k = 5$ parts and at each round, pick one of these parts as a validation dataset and the other remaining parts as the new training dataset. We can now calculate the validation error for a model trained on the new train dataset. The final validation error for a given λ is the average validation error across the $k = 5$ parts. Compute the average cross validation error for

Algorithm 1 Multi-Class Adaboost

:

Inputs: N training samples $\mathbf{X}$, $\mathbf{y}$, Total number of weak classifiers T.
Initialization: Sample weights $\mathbf{D} = [d_1, \ldots, d_N]$ s.t. $d_i = 1/N$ for all $i \in \{1, \ldots, N\}$.
for $m = 1$ to T **do**
    Train a decision tree classifier h_m with maximum depth 4 using training samples $(\mathbf{X}, \mathbf{y})$ and $sample\_weight = \mathbf{D}$
    Compute $err[m] \leftarrow \sum_{i=1}^{N} d_i \mathbb{I}\left(y_i \neq h_m\left(\mathbf{x}_i\right)\right)$
    Compute $\alpha_m \leftarrow \log\left(\frac{1-err[m]}{err[m]}\right) + \log\left(k-1\right)$
    Update sample weights: $d_i \leftarrow d_i \exp\left(\alpha_m \mathbb{I}\left(y_i \neq h_m\left(\mathbf{x}_i\right)\right)\right) \qquad \forall i \in \{1, \ldots, N\}$
    Normalize the sample wights $d_i = \frac{d_i}{\sum_{j=1}^{N} d_j} \qquad \forall i \in \{1, \ldots, N\}$
end for
Outputs: For a new feature $\mathbf{x}$, assign a class:

$$J^*\left(\mathbf{x}\right) = \arg \max_{j \in \{0, \ldots, k-1\}} \sum_{m=1}^{T} \alpha_m \mathbb{I}\left(h_m\left(\mathbf{x}\right) = j\right)$$

---

each $\lambda \in \Lambda$. What λ value seems to give the smallest average validation error? Find the test error for the model trained (on the entire train dataset) using this value of λ. As before, train each model using learning rate $1e^{-7}$, `batch_size`=200 and number of iterations 3000.

(i) (**Decision Trees and Adaboost**): In this part, we will implement Adaboost algorithm for Multi-class classification [1]. First, install scikit-learn, using the following command:

```
pip install -U scikit-learn scipy matplotlib
```

Implement the Adaboost algorithm 1. To train a decision tree classifier, you can use the following command:

$tree =$`DecisionTreeClassifier(`$max\_depth = 4$`).fit(X_train, y_train,` $sample\_weight = D$`)`

where maximum depth $max\_depth = 4$ and sample weights $sample\_weight = D$. To predict the labels from the features, you can use:

```
tree.predict(X_test)
```

Plot a figure depicting the test error and the training error in the same plot as a function of the number of classifiers (T). What is the test error for $T = 1$ and the test error for $T = 200$?

References

[1] Hastie, T., Rosset, S., Zhu, J. and Zou, H., 2009. Multi-class adaboost. Statistics and its Interface, 2(3), pp.349-360.

[2] https://github.com/zalandoresearch/fashion-mnist